Adding IEEE 802.15.4 and 6LoWPAN to an Embedded Linux Device

FOSDEM 2017

2017-02-05, Brussels

Stefan Schmidt
stefan@osg.samsung.com
Samsung Open Source Group
Agenda

- Motivation
- Linux-wpan Project
- Hardware
- Configuration
- Communication with RIOT and Contiki
Motivation
IEEE 802.15.4

- IEEE specifications for Low-Rate Wireless Personal Area Networks
- Not only low-rate, but also low-power
- Designed for small sensors to run months/years on battery with the right duty cycle
- 127 bytes MTU and 250 kbit/s
- Sometimes confused with ZigBee as it is used as PHY and MAC layer there
6LoWPAN

- Physical and MAC layer defined by IEEE 802.15.4 from 2003 onwards (latest update from 2015)
- Series of IETF specifications from 2007 onwards (RFCs 4944, 6282, etc)
The Header Size Problem

- Worst-case scenario calculations
- Maximum frame size in IEEE 802.15.4: 127 bytes
- Reduced by the max. frame header (25 bytes): 102 bytes
- Reduced by highest link-layer security (21 bytes): 81 bytes
- Reduced by standard IPv6 header (40 bytes): 41 bytes
- Reduced by standard UDP header (8 bytes): 33 bytes
- This leaves only 33 bytes for actual payload
- The rest of the space is used by headers (~ 3:1 ratio)

| Frame Header (25) | LLSEC (21) | IPv6 Header (40) | UDP | Payload (33) |
The Header Size Problem

- Worst-case scenario calculations
- Maximum frame size in IEEE 802.15.4: 127 bytes
- Reduced by the max. frame header (25 bytes): 102 bytes
- Reduced by highest link-layer security (21 bytes): 81 bytes
- Reduced by standard IPv6 header (40 bytes): 41 bytes
- Reduced by standard UDP header (8 bytes): 33 bytes
- This leaves only 33 bytes for actual payload
- The rest of the space is used by headers (~ 3:1 ratio)
The Header Size Solution

- IPv6 with link-local and UDP on top
- IPHC with NHC for UDP
- The 48 bytes IPv6 + UDP header could in the best cases be reduced to 6 bytes
- That allows for a payload of *75 bytes* (~ 2:3 ratio)
Linux-wpan

- Platforms already running Linux would benefit from native IEEE 802.15.4 and 6LoWPAN subsystems
- IEEE 802.15.4 transceivers can easily be added to existing hardware designs
- Battery powered sensors on the other hand are more likely to run an OS like RIOT or Contiki
- Example 1: Google OnHub AP which already comes with, de-activated, IEEE 802.15.4 hardware
- Example 2: Ci40 Creator board as home IoT hub
Linux-wpan Project
Linux-wpan Project

- IEEE 802.15.4 and 6LoWPAN support in mainline
- Started in 2008 as linux-zigbee project, from 2012 mainline
- New project name to avoid confusion: linux-wpan
- Normal kernel development model
- Patches are posted and reviewed on the mailing list
- Small community: 2 core devs and ~4 additional people for specific drivers
- Linux-wpan mailing list (~93 people), IRC (~29 people)
Current Status

- ieee802154 layer with softMAC driver for various transceivers
- 6LoWPAN with fragmentation and reassembly (RFC 4944)
- Header compression with IPHC and NHC for UDP (RFC 6282), shared with Bluetooth subsystem
- Link Layer Security
- Testing between Linux, RIOT and Contiki
- Mainline 4.1 onwards recommended
- Active development, newer is better :-(
Development Boards

- MIPS based Ci40 Creator (CA-8210)
- Raspberry Pi with Openlabs shield (AT86RF233)
- Transceivers can be hooked up via SPI (all drivers have devicetree bindings)
- ATUSB dongle
Hardware
Hardware Requirements

- Free SPI port
- Some additional free GPIO pins
- You can choose between off the shelf modules or adding it directly to your design
- Alternatively you could use USB
Devicetree Bindings

- Boards need devicetree support
- All our drivers have bindings
- Example for the at86rf233:

```device-tree
&spi {
    status = "okay";
    at86rf233@0 {
        compatible = "atmel,at86rf233";
        spi-max-frequency = <6000000>;
        reg = <0>;
        interrupts = <23 4>;
        interrupt-parent = <&gpio>;
        reset-gpio = <&gpio 24 1>;
        sleep-gpio = <&gpio 25 1>;
        xtal-trim = /bits/ 8 <0x0F>;
    }
};
```
Hardware Support

- Mainline drivers for at86rf2xx, mrf24j40, cc2520, atusb and adf7242
- Pending driver for ca-8210
- Old out of tree driver for Xbee
- ATUSB dongle to be used on your workstation
## Transceiver Comparison

<table>
<thead>
<tr>
<th>Chipset</th>
<th>Interface</th>
<th>Driver</th>
<th>2.4 GHz</th>
<th>Sub GHz</th>
<th>ARET</th>
<th>IEEE specs</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADF7242</td>
<td>SPI or PMOD</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>2003-2006</td>
</tr>
<tr>
<td>AT86RF212</td>
<td>SPI + GPIO</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>2003-2006</td>
</tr>
<tr>
<td>AT86RF212b</td>
<td>SPI + GPIO</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>2003-2011</td>
</tr>
<tr>
<td>AT86RF215</td>
<td>SPI + GPIO + LVDS</td>
<td>✘</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>2003-2011, 15.4g</td>
</tr>
<tr>
<td>AT86RF230</td>
<td>SPI + GPIO</td>
<td>✘</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>2003</td>
</tr>
<tr>
<td>AT86RF231</td>
<td>SPI + GPIO</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>2003-2006</td>
</tr>
<tr>
<td>AT86RF233</td>
<td>SPI + GPIO</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>2003-2011</td>
</tr>
<tr>
<td>ATUSB (AT86RF231)</td>
<td>USB</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>2003-2006</td>
</tr>
<tr>
<td>RZUSB (AT86RF230)</td>
<td>USB</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>✔️*</td>
</tr>
<tr>
<td>CA8210</td>
<td>SPI + GPIO</td>
<td>✔️*</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>2003-2006</td>
</tr>
<tr>
<td>CC2420</td>
<td>SPI + GPIO</td>
<td>✘</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>2003</td>
</tr>
<tr>
<td>CC2520</td>
<td>SPI + GPIO</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✘</td>
<td>2003-2006</td>
</tr>
<tr>
<td>CC2531</td>
<td>USB</td>
<td>✘</td>
<td>✔️</td>
<td>✘</td>
<td>✘*</td>
<td>2003-2006</td>
</tr>
<tr>
<td>MRF24J40</td>
<td>SPI + GPIO</td>
<td>✔️</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>2003</td>
</tr>
<tr>
<td>XBee</td>
<td>UART</td>
<td>✘*</td>
<td>✔️</td>
<td>✘</td>
<td>✔️</td>
<td>2003??</td>
</tr>
</tbody>
</table>
Virtual Driver

- Fake loopback driver (similar to hwsim of wireless)
- Great for testing
- Support for RIOT and OpenThread to use this when running as native Linux process
- Will help interoperation testing between the different network stacks in an virtual environment

$ modprobe fakelb numlbs=4

$ Configure for Linux, RIOT, OpenThread and monitor
Configuration
Wpan-tools: iwpan

- Userspace configuration utility
- Netlink interface ideas as well as code borrowed from the iw utility
- Used to configure PHY and MAC layer parameters
- Including: channel, PAN ID, power settings, short address, frame retries, etc
- Packaged by some distributions (Fedora and Debian up to date, Ubuntu on 0.5, OpenSUSE, Gentoo, Arch, etc missing)
Wpan-tools: wpan-ping

- Ping utility on the IEEE 802.15.4 layer
- Not a full ICMP ping replacement, but good enough for some basic testing and measurements

# run on server side
$ wpan-ping --daemon

# run on client side
Interface Bringup

- The wpan0 interface shows up automatically
- Setting up the basic parameters:
  
  
  ```
  $ ip link set lowpan0 down
  $ ip link set wpan0 down
  $ iwpan dev wpan0 set pan_id 0xabcd
  $ iwpan phy phy0 set channel 0 26
  $ ip link add link wpan0 name lowpan0 type lowpan
  $ ip link set wpan0 up
  $ ip link set lowpan0 up
  ```
Monitoring

- Setting up the interface in promiscuous mode:
  $ iwpan dev wpan0 del
  $ iwpan phy phy0 interface add monitor%d type monitor
  $ iwpan phy phy0 set channel 0 26
  $ ip link set monitor0 up
  $ wireshark -i monitor0

- No automatic channel hopping (you can change the channel manually in the background)
Communication with RIOT & Contiki
RIOT

- “The friendly Operating System for the Internet of Things” (LGPL)
- Testing against Linux-wpan is part of the release testing process for RIOT
- Active developer discussions and bug fixing between projects
Contiki

- “The Open Source OS for the Internet of Things” (BSD)
- Very fragmented project
- Sadly many forks for academic or commercial purpose which have a hard time to get merged
- Still an important role as IoT OS for tiny devices
## Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Linux</th>
<th>RIOT</th>
<th>Contiki</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE 802.15.4: data and ACK frames</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>IEEE 802.15.4: beacon and MAC command frames</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>IEEE 802.15.4: scanning, joining, PAN coordinator</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>IEEE 802.15.4: link layer security</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>6LoWPAN: frame encapsulation, fragmentation, addressing (RFC 4944)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>6LoWPAN: IP header compression (RFC 6282)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>6LoWPAN: next header compression, UDP only (RFC 6282)</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>6LoWPAN: generic header compression (RFC 7400)</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>6LoWPAN: neighbour discovery optimizations (RFC 6775)</td>
<td>Partial</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Mesh link establishment draft</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
</tbody>
</table>
Others

- mbed OS from ARM: network stack is closed source, so nothing to test against
- Zephyr: new network stack not tested yet
- OpenThread: Open Source implementation of the Thread protocol
Future
Linux-wpan Future

- Implement missing parts of the IEEE 802.15.4 specification
  - Beacon and MAC command frame support
  - Coordinator support in MAC layer and wpan-tools
  - Scanning
- Add better support for HardMAC transceivers
- Neighbour Discovery Optimizations (RFC 6775), started
- Evaluate running OpenThread on top of linux-wpan
- Configuration interface for various header compression modules
- Expose information for route-over and mesh-under protocols
Summary
Take away

• Running an IEEE 802.15.4 wireless network under Linux is not hard
• Tooling and kernel support is already there
• Border router scenario most likely use case but nodes or routers also possible
Thank you!

http://www.slideshare.net/SamsungOSG
References

- IEEE 802.15.4 specification (PHY and MAC layer)
  http://standards.ieee.org/about/get/802/802.15.html
- RFC 4944: Transmission of IPv6 Packets over IEEE 802.15.4 Networks
- RFC 6282: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks
- RFC 7400: 6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)
- Linux-wpan source and project pages
  https://github.com/linux-wpan
  http://wpan.cakelab.org/
6LoWPAN Fragmentation

- IPv6 requires the link to allow for a MTU of at least 1280 bytes
- This is impossible to handle in the 127 bytes MTU of IEEE 802.15.4
- 6LoWPAN 11 bit fragmentation header allows for 2048 bytes packet size with fragmentation
- But fragmentation can still lead to bad performance in lossy networks, best to avoid it in the first place
IPv6 Header Compression (IPHC)

- Defining some default values in IPv6 header
  - Version == 6, traffic class & flow-label == 0, hop-limit only well-known values (1, 64, 255)
  - Remove the payload length (available in 6LoWPAN fragment header or data-link header)
- IPv6 stateless address auto configuration based on L2 address
  - Omit the IPv6 prefix (global known by network, link-local defined by compression (FE80::/64)
  - Extended: EUI-64 L2 address use as is
  - Short: pseudo 48 bit address based short address: PAN_ID:16 bit zero:SHORT_ADDRESS

<table>
<thead>
<tr>
<th>Version</th>
<th>Traffic Class</th>
<th>Flow Label (20 bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Payload Length (16 bit)</th>
<th>Next Header</th>
<th>Hop Limit (8 bit)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source Address (128 bit)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Destination Address (128 bit)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>6LoWPAN Header IPHC link-local (2 bytes)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Dispatch</th>
<th>LoWPAN_IPHC</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>6LoWPAN Header IPHC multi-hop (7 bytes)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Dispatch</th>
<th>LoWPAN_IPHC</th>
<th>Hop Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source Address</th>
<th>Destination Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Next Header Compression

- NHC IPv6 Extension Header compression (RFC6282)
  - Hop-by-Hop, Routing Header, Fragment Header, Destination Options Header, Mobility Header
- NHC UDP Header compression (RFC6282)
  - Compressing ports range to 4 bits
  - Allows to omit the UDP checksum for cases where upper layers handle message integrity checks
- GHC: LZ-77 style compression with byte codes (RFC7400)
  - Appending zeroes, back referencing to a static dictionary and copy
  - Useful for DTLS or RPL (addresses elided from dictionary)
Link Layer Security
Link Layer Security

- Specified by IEEE 802.15.4
- It defines confidentiality (AES-CTR), integrity (AES CBC-MAC) and encryption and authentication (AES CCM) security suites
- Key handling, key exchange, roll over, etc is not defined
- Tested Linux against Linux and Contiki 3.0
- No way to test against RIOT as they have no LLSEC support right now
LLSEC Linux-wpan

- Needs the llsec branch in wpan-tools for configuration

- `CONFIG_IEEE802154_NL802154_EXPERIMENTAL`

  - `$ iwpan dev wpan0 set security 1`
  - `$ iwpan dev wpan0 key add 2 $KEY 0 $PANID 3 $EXTADDR`
  - `$ iwpan dev wpan0 seclevel add 0xff 2 0`
  - `$ iwpan dev wpan0 device add 0 $PANID $SHORTADDR $EXTADDR 0 0`
LLSEC Contiki 3.0

- You need the following Contiki build options configured in your project-conf.h to make use of LLSEC with network wide key:

  ```
  #define NETSTACK_CONF_LLSEC noncoresec_driver
  #define LLSEC802154_CONF_SECURITY_LEVEL FRAME802154_SECURITY_LEVEL_ENC_MIC_32

  #define NONCORESEC_CONF_KEY {
      0x00, 0x01, 0x02, 0x03, \
      0x04, 0x05, 0x06, 0x07, \
      0x08, 0x09, 0x0A, 0x0B, \
      0x0C, 0x0D, 0x0E, 0x0F, \
  }
  ```